



GrabberBot—Program It

During the programming of the GrabberBot, I'm not going to tell you to use any particular method for programming—I'm leaving that up to you. You might choose to open up the NXT software and start dropping blocks until you've got the program completed; then you'll upload the program, test it, and debug it if you find any problems. Or you might use the method described in Chapter 12, moving forward with your program only after you've successfully placed a block and then tested that the bot does what it's supposed to do.

Or you can follow along with the method I'll present in this chapter, which is a slight variation on the second method. I'll place a block, configure it, upload it to my bot, and then test it. If the bot doesn't perform as anticipated, I'll reexamine the block and its configuration settings and see whether I've made a mistake. In some instances, such as the placement of a WAIT block, I might place two blocks and then test—the WAIT blocks are sometimes so simple I choose to test them at the same time as a more advanced block.

Whichever method you choose, take notes of your successes, but **ESPECIALLY** your failures. Try to determine where you made your mistakes—were you in a hurry and missed something important? Did you choose the wrong programming block for the job? If you pay attention to your mistakes and learn why you made them and how you solved them, your bot programming skills will definitely improve.

Down the Tunnel . . . Again . . .

If you're following along with the method I've chosen, open up the Lego Mindstorms NXT software, type **GrabberBot** into the blank text field labeled **Start New Program**, and then click the **Go** button (see Figure 16-1).



Figure 16-1. Enter *GrabberBot* for the program name and click *Go*.

Note If the RoboCenter area is visible on the far right of your screen, click the small red X in the upper-right corner of the software, and it will free up more visible workspace on your screen.

Just like with the SnapShotBot in Chapter 11, I have placed the Touch Sensor on the back of the bot to use as a **Start** button for the GrabberBot. I've found that if I don't need the Touch Sensor for a bot design, I can include it as the **Start** button, and it keeps me from having to press any buttons on the Brick—and sometimes the location or orientation of the Brick can make pressing its buttons difficult. By placing the Touch Sensor in an easy-to-reach location, I don't have to worry about accidentally bumping my bot when I pull my hand away quickly after pressing the orange **Run** button. (Some programmers put a 5- or 10-second WAIT block at the beginning of their programs so they can press the orange **Run** button on the Brick and still have time to place their bot correctly before it begins to run—that works, too.)

The first block I'll place in my program (again) is a LOOP block (see Figure 16-2) configured to detect the press and release of the Touch Sensor. When the button is pressed and released, the loop will break and the remaining programming blocks will begin to run.

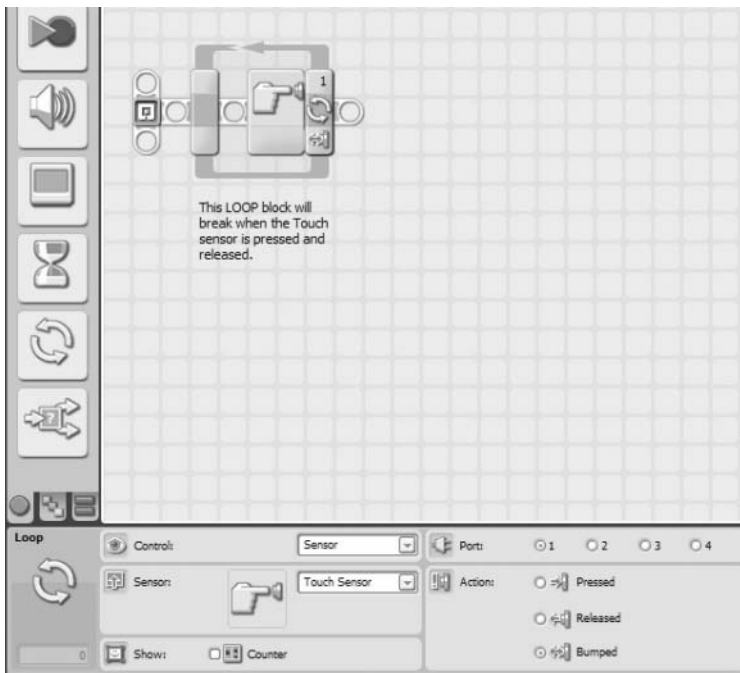


Figure 16-2. A LOOP block and Touch Sensor trigger are used to start the GrabberBot.

Look back at my Task List (turn back to Figure 14-2). My first task is “Move toward the end of the tunnel.” There are actually two movements here, though. The first is the bot moving toward the scroll and stopping a safe distance away to begin detecting the wall. The second movement is toward the scroll (at a slower pace) and letting the Ultrasonic Sensor detect the wall and the proper place for the GrabberBot to stop to grab the scroll.

Right now, I need to perform the first movement, but I still haven't told you how I plan on programming the bot to keep from moving too far (and accidentally bumping the scroll). Look at Figure 16-3.

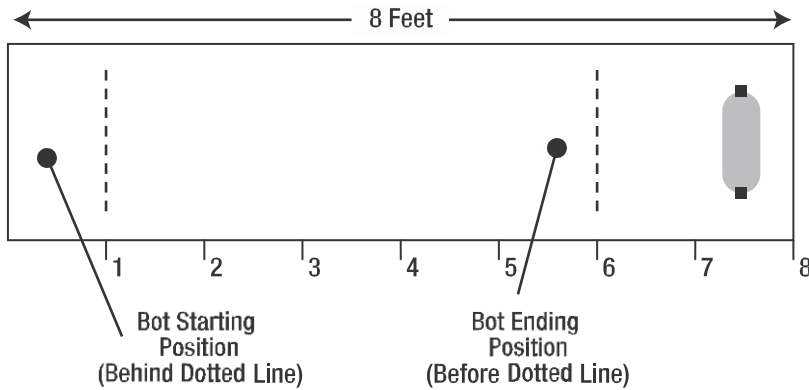


Figure 16-3. *Determining the correct distance for the GrabberBot*

In this figure, I've labeled the GrabberBot's starting position as being behind an imaginary line that is at the 1-foot mark inside the tunnel. My bot is slightly longer than 1 foot, but my calculations will be based on the FRONT wheels being behind this line. The tunnel is 8 feet deep, and the scroll is 4 inches from the back wall of the tunnel. I have placed another imaginary line at the 6-foot mark. If I can program my bot to stop before it reaches the 6-foot point shown in Figure 16-3, it should be a safe enough distance away not to accidentally bump the scroll.

The method I'm going to use for accomplishing this is more exact than the method you saw for the SnapShotBot in Chapter 12. In Chapter 10, I divided distances to travel by the circumference of the wheel (7 inches) and rounded the numbers up or down. The distances the SnapShotBot traveled weren't as exact as I need for my GrabberBot to travel. So I need a different method for fine-tuning the distances my GrabberBot will move. There's a little math involved (sorry), but I don't think you'll have trouble following along—but do grab a calculator. (Unlike your school teacher, I allow calculators!)

Let's start off with a simple observation. The wheels of your Grabberbot are 7 inches in circumference. This means that when the wheel makes one complete rotation on the ground, it will have traveled 7 inches. Are you with me so far? Now it just so happens that when a wheel spins completely around for 1 rotation, it has also spun 360 degrees. When you travel around a circle and return to your starting point, you have made a trip of 360 degrees, got it? Okay, so if the wheel spins for half a rotation, how many degrees has it traveled? Did you answer 180 degrees? What about a quarter of a rotation? The answer is 90 degrees.

Now, let's move our bot around a little more. If your bot moves forward for 8 rotations, how many degrees has it traveled? To find the answer, you simply multiple 360 (degrees) by 8 (number of rotations) and the answer is 2880. If you wanted to configure a MOVE block to spin your robot for 8 rotations, you could also program it to spin for 2880 degrees—it will move the same distance. Test it!

And the best part about converting rotations to degrees is that you don't have to round your number of rotations off—if you want your bot to move 2.25 rotations, you simply multiply 360 by 2.25 and the answer is 810 degrees!

And the math works both ways, by the way. If you have a bot you want to move forward 3432 degrees, how can you determine how many rotations to program it? Just do the reverse—divide the number of degrees of movement by 360, and the answer will be the number of rotations you need to program your bot to move. So, 3432 degrees divided by 360 equals 9.5 rotations (round your numbers to one decimal place, but don't worry—for most of your bots, this should be accurate enough).

So, before moving forward, let's make sure you understand two simple rules:

1. To convert rotations to degrees, simply *multiply* the number of rotations by 360, and the answer is the equivalent number of degrees.
2. To convert degrees to rotations, simply *divide* the number of degrees by 360, and the answer is the equivalent number of rotations.

So, how will I use this information to guide my GrabberBot to the proper location? Well, if the GrabberBot wants to move from the 1-foot mark to the 6-foot mark, it will need to move *NO MORE THAN* 5 feet, right?

I know that 5 feet is the equivalent of 60 inches (there are 12 inches in 1 foot, so for 5 feet, I simply multiply 5 by 12 to get 60). Since one wheel rotation is 7 inches, if I divide 60 inches by 7 inches, this will give me the number of rotations a wheel will spin if it travels 5 feet. The answer is 8.6 rotations. Now comes the math—Rule #1 tells me all I need to do to convert 8.6 rotations to degrees is multiply it by 360. When I do this, I get an answer of 3096 degrees. When I program my first MOVE block, I will tell my GrabberBot to move forward 3096 degrees, and I know that it should be stopped on or behind that second imaginary line in Figure 16-3.

Before I continue, are you now aware that robotics does involve some mathematics? Would you believe that more advanced robots will require even more advanced math skills? Since I let you use a calculator, please allow me to encourage you to not ignore your math and science studies, okay? If you like designing robots and hope your way up to more advanced designs, you'll need to concentrate in school on ALL your subjects (including History)! Okay, enough lecturing—back to the programming.

So now I can place that first MOVE block (see Figure 16-4). Notice I've configured it to spin 3096 degrees, which is equivalent to 5 feet.

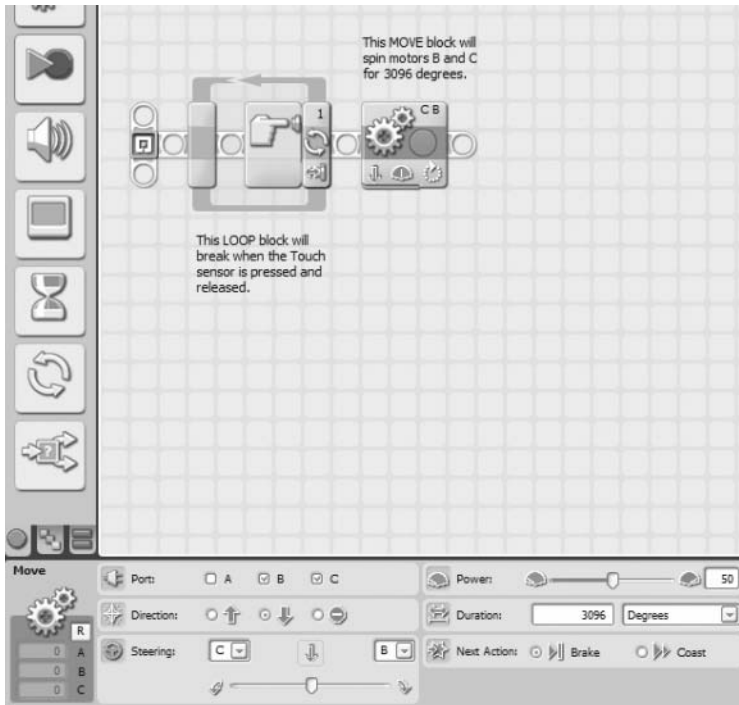


Figure 16-4. This MOVE block controls the GrabberBot's movement down the tunnel.

What if my calculations were a little off? Well, I did round the number of rotations from 8.5714 (when I divided 60 inches by 7 inches earlier), so could this make my GrabberBot possibly bump the scroll? If you look again at Figure 16-3, you'll notice that I've got another foot-and-a-half of distance before the scroll, so I should be safe. That's why I intentionally chose to stop the bot at the 6-foot mark and not the 7-foot position.

Now I'm going to upload my program and test it. I have a measuring tape, and I want to make sure that my bot moves forward about 5 feet and no more. I test it and . . . it works. I press the **Start** button (Touch Sensor), and the bot moves forward almost exactly 5 feet from its starting point. Excellent.

Because I need to take a deep breath at this point (and maybe you do, too), I'm going to have my bot pause for 10 seconds. I'm doing this because as I mentioned in Chapter 14, if I find that my bot isn't lined up exactly where I want it, I could use the string that I'm going to tie to the rear of my bot and pull it back. It's not the best solution, but I really want my bot to make one attempt on the scroll, and I want it to be successful. So I'll add in a WAIT block, which will let me look down the tunnel and make certain the bot is pointed directly forward and that the arm mechanism will fit smoothly underneath the scroll's supporting legs (see Figure 16-5).

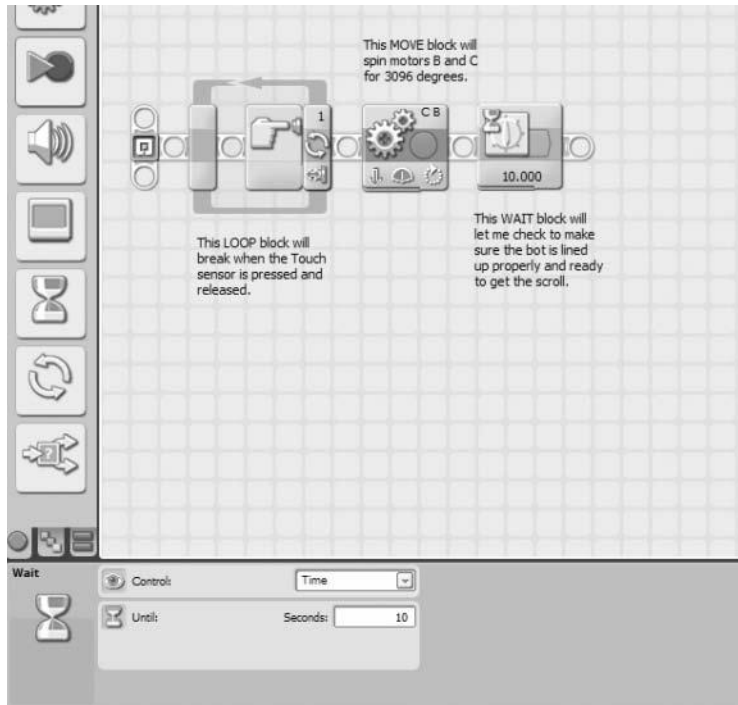


Figure 16-5. A WAIT block will let me check to make sure the GrabberBot is ready to get the scroll.

I'll tie a string to the rear of my bot, but hopefully I won't have to use it. The WAIT block is just a safety precaution that I've decided to add—feel free to leave it out if you feel your bot is ready to move forward and grab the scroll.

Approaching the Scroll

Now, the next movement for my GrabberBot is a little trickier. I've got to have the bot crawl forward, very slowly, and try to detect the back wall. Why the back wall, you may ask? Well, my GrabberBot is designed so that its arm mechanism will come in UNDER the scroll and lift up. It's small and I've designed it to fit between the two support legs. What I do NOT know is how close I need for the Ultrasonic Sensor to be to the back wall. If you take a look at my final GrabberBot design (refer back to Figure 15-55), you'll notice that the Ultrasonic Sensor sticks out a little further than the arm mechanism. What I'll do is test this over and over until I find the best position for the Ultrasonic Sensor to trigger—this position will be when the lifting arm is directly under the scroll, the best position for it to get a good lift-off of the scroll. It could be 4 inches from the wall or less than 1 inch—I won't know until I test it.

And since I need to test it, this is probably the best place for me to explain my test environment, which is shown in Figure 16-6.



Figure 16-6. *Photograph of the “scroll” and support legs*

What I’ve done for my test environment is measure out 8 feet from a wall in my living room. I place some obstacles to represent the tunnel walls (okay, they are couch cushions—use what you have). I measure 4 inches from the wall and place two small tin cans about 6 inches apart—these are the support legs for my scroll. And finally, for the scroll, I sit on top of the cans a small, fuzzy paint roller. Simple, but it all works great.

Now, before I begin testing, I need to program my bot to slowly approach the scroll. To do this, I’m going to use a LOOP block that will break when the Ultrasonic Sensor is triggered. So I first drop in that LOOP block (see Figure 16-7).

For now, I’ve configured the Ultrasonic Sensor to trigger when it gets about 8 inches from the back wall (or other obstacle).

Next, I’ll drop in a MOVE block (see Figure 16-8).

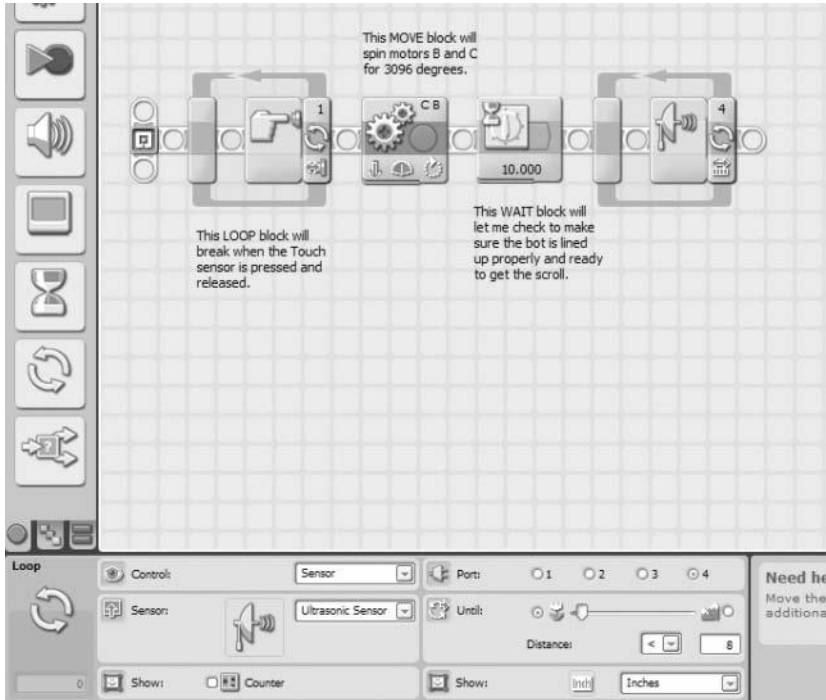


Figure 16-7. The LOOP block will break when the Ultrasonic Sensor is triggered.

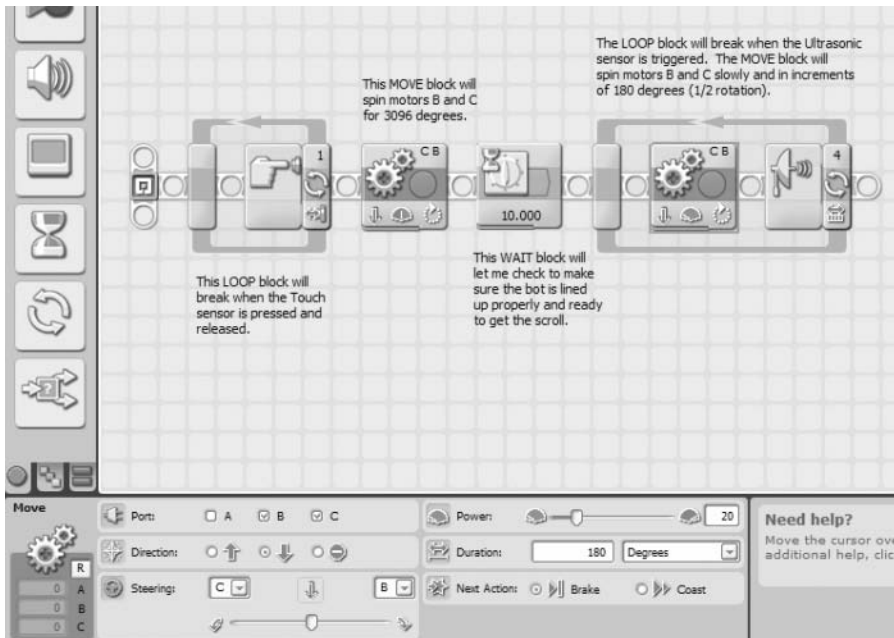


Figure 16-8. The MOVE block will move the bot forward slowly for a short distance.

I'm going to have the motors B and C spin slowly (**Power** is set to 20), and a just a little bit at a time. The bot will move forward a short distance (.5 rotation or 180 degrees), and the Ultrasonic Sensor will see whether the wall is within 8 inches. If not, the bot will move forward again. It will keep doing this until the Ultrasonic Sensor breaks the loop.

Finally, I want to add a WAIT block to allow the bot to fully stop (see Figure 16-9). It gives me time to pull the bot back with the string if I discover the bot is approaching the scroll at an incorrect angle.

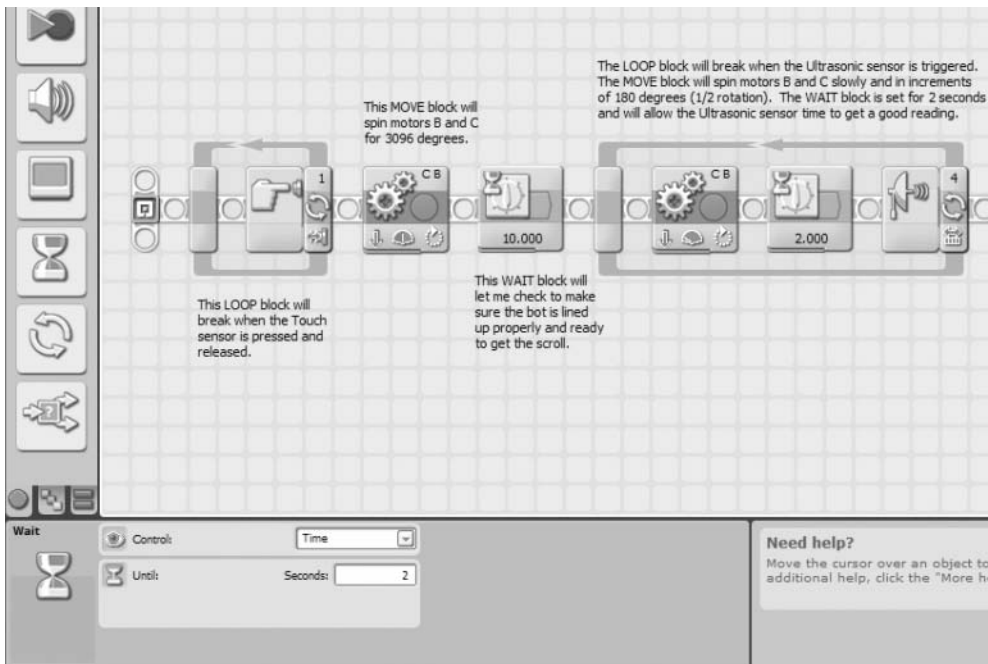


Figure 16-9. The WAIT block allows the Ultrasonic Sensor to get a good reading.

And now I upload the program and test. Below are my results—it took me four tests to determine the proper configuration of the Ultrasonic Sensor:

Test 1: Ultrasonic Sensor configured for < 8 inches—arm not under the scroll.

Test 2: Ultrasonic Sensor configured for < 6 inches—arm almost under the scroll.

Test 3: Ultrasonic Sensor configured for < 5 inches—arm almost but not completely under the scroll.

Test 4: Ultrasonic Sensor configured for < 4 inches—arm underneath the scroll.

■ **Note** Every Ultrasonic Sensor is different, and your Ultrasonic Sensor will have a slightly different sensitivity than my Ultrasonic Sensor. Because of this, you need to test your bot and record your own results. The final value for my Ultrasonic Sensor (< 4 inches) may or may not work with your GrabberBot! Although the Ultrasonic Sensor is fairly accurate, always test with your own values, not the ones I am using—just to be safe!

At this point, my bot will now move down the tunnel and stop about 2 feet in front of the scroll. It then begins to move slowly toward the scroll (stopping and pausing every 3 inches) until the Ultrasonic Sensor detects the wall and triggers the bot to stop. Now my little bot's lifting arm is directly under the scroll and ready to lift it up and return it to me.

Acquiring the Scroll

When my GrabberBot lifts the scroll off its support legs, I intend for it to pin the scroll against the beams above the arm mechanism (flip back to Figure 15-55). By holding the scroll against these beams, it is less likely that the scroll will roll or fall out of the arm mechanism on its return trip.

Now, to do this, I will need motor A to apply continuous upward pressure against the scroll. To do this, I will simply put a MOVE block into a LOOP set to loop Forever. But if I have the LOOP block configured this way, it will never break. How will the bot be given instructions to return with the scroll?

The solution to this involves *parallel processes*. What are parallel processes? They are simply programming commands (blocks) your bot will execute simultaneously. For example, you could program a bot with a MOVE block that has motors B and C moving the bot randomly around the room, never stopping. At the same time, you could program the bot to also have motor A rotating the Sound Sensor back and forth, listening for sound input. So far, all of the bots I've shown you how to program involve one block running at a time (like a WAIT block or a LOOP block). When one block finishes, the next block starts.

In order to program my bot to do two things at once, I will create another path (beam) where I'll place programming blocks that will run simultaneously. But before I do this, I'll finish the program as if motor A already has the scroll pinned. And when I'm finished, I will then go back and add in that second sequence beam that will contain the blocks for motor A.

Okay, so if I assume that my GrabberBot already has the scroll pinned, all that's left is for the bot to move backward and return to the end of the tunnel. I'll do this by adding in a MOVE block for motors B and C (see Figure 16-10). I don't want the bot to return too quickly, so I'm also going to program it for a slower speed (**Power** set to 40).

Now I upload the program and verify that the bot will reverse direction. The bot worked fine, so now I'm going to add in the necessary blocks for the bot to grab the scroll—for this I'll be adding a parallel process as mentioned earlier.

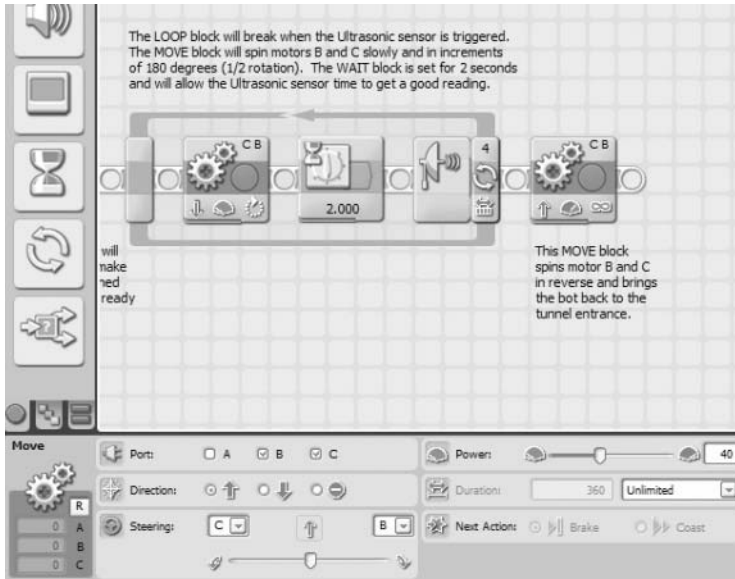


Figure 16-10. A MOVE block is used to bring the bot back to the tunnel entrance.

Fortunately, the NXT programming language allows us to do this by simply creating another beam that breaks off of the main beam. To do this, simply hold down the Shift key (the mouse pointer will change to a funny little symbol) and click where you want this new beam to start. Then drag up (or down) and you'll see a new beam starting. When you click once, you can make a bend in the beam and move it in a new direction. Double-click where you want to place the end of the new beam. I want the parallel action to occur at the same time as the MOVE block that returns the bot to the tunnel entrance. So I have to start this new beam just before that MOVE block's position (see Figure 16-11).

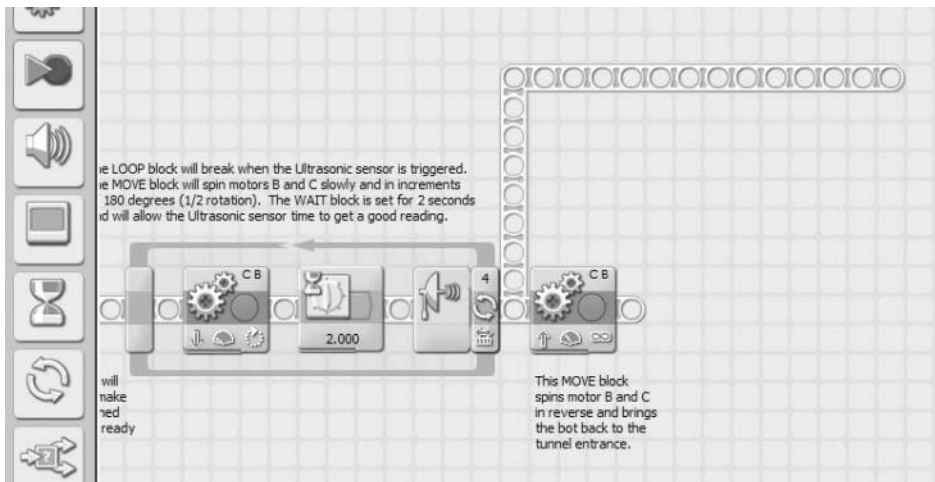


Figure 16-11. Place the start of the new beam before the MOVE block.

As you can see in Figure 16-11, I've dragged a new beam up and over (to the right) to allow me some space to place new blocks. I can still place blocks on the original beam, as you'll see later in Figure 16-13.

At this point in the program, my bot's arm is under the scroll, and I want it to lift up just enough to pin the scroll against the upper beams of my GrabberBot. I'll need to test motor A many times to determine the proper number of rotations or degrees to do this. But for now, I'll place a MOVE block to perform the lifting (see Figure 16-12). I configure it to lift slowly (**Power** is set to 30) and for an initial 180 degrees.

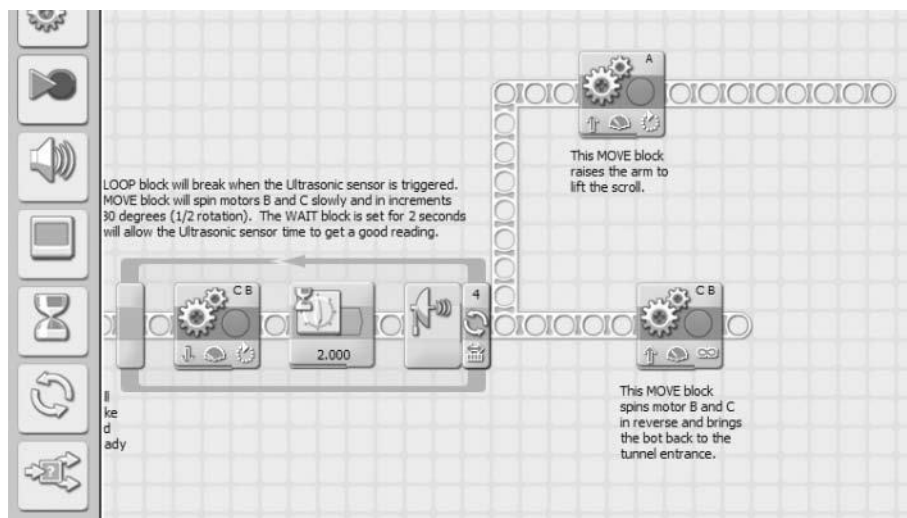


Figure 16-12. The MOVE block will spin motor A to lift the scroll.

Now I upload and test the program. And I encounter three problems. The first problem is that the 180 degrees I've configured motor A to spin is not enough (the scroll's weight may be the issue). I test it again and am able to determine that 270 degrees is the best setting to pin the scroll against the upper beams.

The second problem requires me to increase the **Power** setting from 30 to 60—I need this additional power to lift the weight of the scroll. Remember, TEST OFTEN!

The third problem requires a short explanation. When the program runs and the extra beam splits off (see Figure 16-11), the program will execute the first block in the upper beam and the first block in the original beam at the same time. What this means is that motor A will lift (the upper, new beam) at the same time as my MOVE block (in the lower, original beam) has motors B and C spin in reverse to return the bot. *Big problem!* The bot needs to first grab the scroll and then move in reverse.

To fix this, I'll simply add a WAIT block that will run at the same time as motor A is lifting. I do this because I want to give the bot time to lift the scroll and pin it before the bot begins to move. I'll put in a WAIT block configured for 5 seconds as shown in Figure 16-13 and another WAIT block configured for 10 seconds as shown in Figure 16-14. The 10-second WAIT block lets the robot pause before the lifting operation. The 5-second WAIT block allows the arm to close before the "reverse" MOVE block is executed and the robot returns with the scroll.

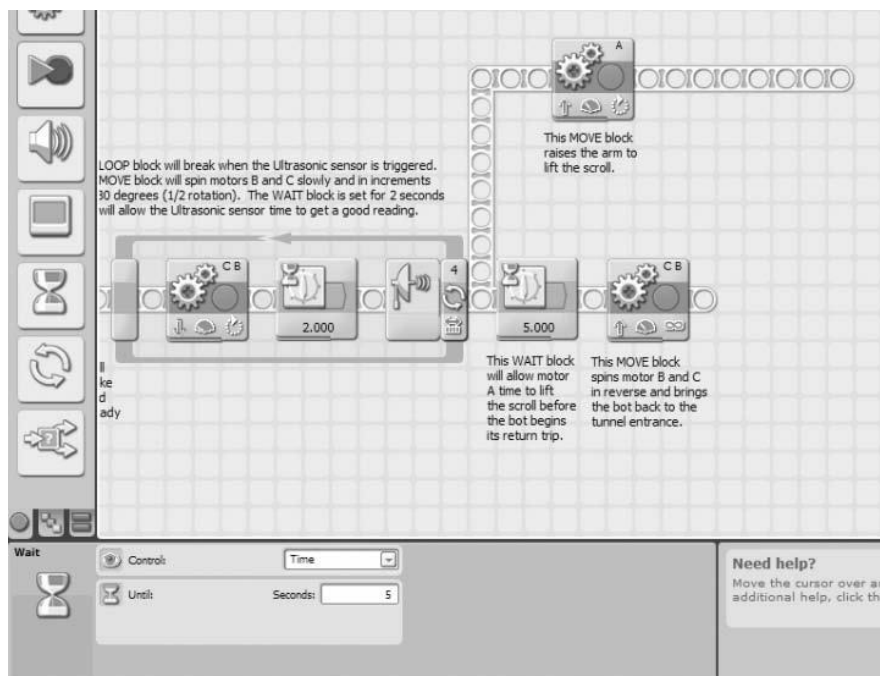


Figure 16-13. *The WAIT block will give motor A time to lift the scroll.*

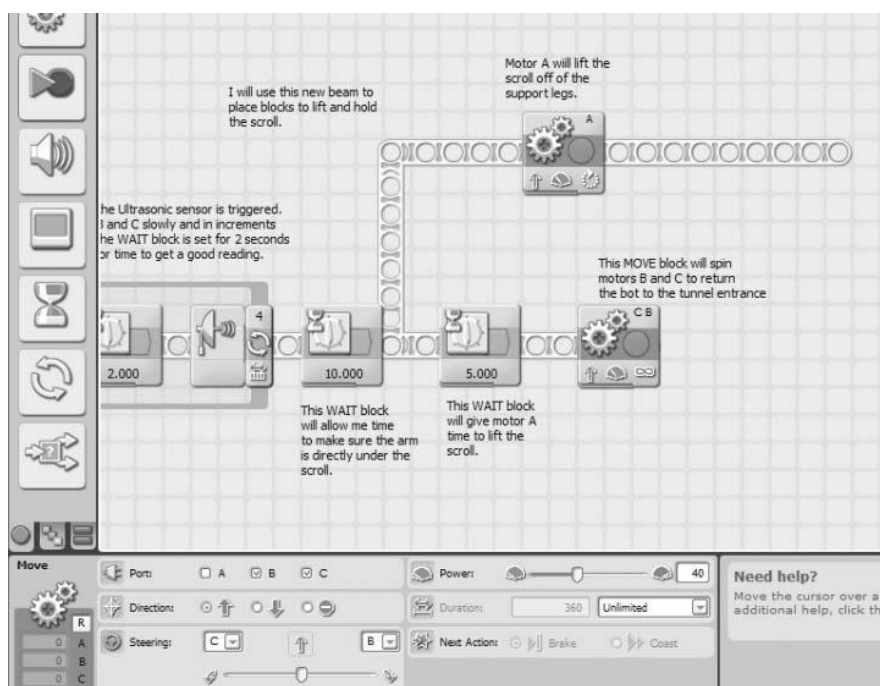


Figure 16-14. An additional WAIT block allows the robot to get everything ready to move.

And once again, I test my bot. Because I've been testing my bot often, usually after each major programming block is added, my bot doesn't require any additional changes. On its *VERY FIRST TEST*, this is what happens:

1. I push the **Start** button (Touch Sensor).
2. The bot moves forward toward the scroll about 5 feet and stops. It doesn't touch the tunnel sides at all (okay, cushions from my couch).
3. The bot moves 3 inches forward, the Ultrasonic Sensor tries to detect the wall and fails, and the bot moves forward another 3 inches.
4. The bot repeats step 3 about 6 times before the Ultrasonic Sensor is triggered.
5. Motor A spins, and the lifting arm raises the scroll and pins it against the upper beams of the GrabberBot.
6. A few seconds later, the bot begins moving backward and brings the scroll back to me.

PERFECT TEST!

Just to be sure, I perform two more tests with the GrabberBot, and they're successful. With three successful runs of the GrabberBot, I am fairly confident in my bot's ability to run down the tunnel, find the scroll, lift it off the support legs, and then return it to me.

Take your GrabberBot and test it until you are able to successfully retrieve the test scroll a few times. When you're ready, set up the scroll one final time and run the bot. If it works, congratulations!

You've got the scroll, and the team is one step closer to finding King Ixtua's burial chamber . . .